UNITED STATES PATENT AND TRADEMARK OFFICE

[illegible]

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/637,169	08/08/2003	Marc Tremblay	SUN-P9325-MEG	2951

57960 7590 03/08/2007
SUN MICROSYSTEMS INC.
C/O PARK, VAUGHAN & FLEMING LLP
2820 FIFTH STREET
DAVIS, CA 95618-7759

EXAMINER	
FENNEMA, ROBERT E	
ART UNIT	PAPER NUMBER
2183	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	03/08/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No.	Applicant(s)
	10/637,169	TREMBLAY ET AL.
	Examiner	Art Unit
	Robert E. Fennema	2183

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1) ☒ Responsive to communication(s) filed on 29 January 2007.

2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.

3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4) ☒ Claim(s) 1-4,6-16 and 18-25 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5) ☐ Claim(s) _____ is/are allowed.

6) ☒ Claim(s) 1-4,6-16 and 18-25 is/are rejected.

7) ☐ Claim(s) _____ is/are objected to.

8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9) ☐ The specification is objected to by the Examiner.

10) ☐ The drawing(s) filed on _____ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a) ☐ All b) ☐ Some * c) ☐ None of:

1. ☐ Certified copies of the priority documents have been received.
2. ☐ Certified copies of the priority documents have been received in Application No. _____.
3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date 12/5/06;1/29/07.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-4, 6-16, and 18-25 have been considered. Claims 1, 7, 13, 19, and 25 have been amended as per Applicants request.

Double Patenting

2. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. A nonstatutory obviousness-type double patenting rejection is appropriate where the conflicting claims are not identical, but at least one examined application claim is not patentably distinct from the reference claim(s) because the examined application claim is either anticipated by, or would have been obvious over, the reference claim(s). See, e.g., *In re Berg*, 140 F.3d 1428, 46 USPQ2d 1226 (Fed. Cir. 1998); *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970); and *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) or 1.321(d) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent either is shown to be commonly owned with this application, or claims an invention made as a result of activities undertaken within the scope of a joint research agreement.

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

3. Claims 1-4, 6-16, and 18-25 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claims 1, 3, 5-13, 15, and 17-25 of copending Application No. 10/637,166 in view of Rajwar et al. and Moss et al. The copending Application teaches executing a start transactional execution instruction to mark the beginning of a block of instructions to be executed transactionally, where changes are not committed unless it successfully completes, but

does not teach a fail instruction, where if encountered, terminates the execution without committing results of the execution, retrying the execution a specified number of times, and obtaining a lock after a specified number of tries.

Moss teaches a variety of primitives to implement transactional execution, including an abort instruction, which causes the machine to discard all updates to the write set. This instruction is combined with a commit instruction and a validate instruction, with the disclosed advantages of allowing a user to define customized regions for transactional execution. However, neither of these references teach retrying the execution a specified number of times, and obtaining a lock if the fail instruction continues to be encountered (presumably after the specified number of times, which is not entirely clear from applicant's claim language), rather they will either try again (possibly forever without success), or will execute an abort instruction which will send the user to some kind of error handler, neither of which guarantees execution. However, Rajwar teaches a multithreaded system which implements critical sections, similar to Moss's and the copending Application's systems, but when the system can not enter the critical section, it tries again several times, and if it continues to fail, will acquire a lock in order to guarantee forward progress (Page 297, Section 3.2). This is advantageous if the user does not wish to use an abort instruction, and wants or needs to execute the instruction, by forcing the system to acquire a lock, it prevents a deadlock which would cripple processor performance. Given these advantages, one of ordinary skill in the art at the time the invention was made would have been motivated to include Rajwar's teachings of acquiring a lock on the critical section after a set number of retries, in order to guarantee forward progress of the code.

This is a provisional obviousness-type double patenting rejection.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 1-4, 6-16, and 18-25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Moss et al ("Transactional Memory: Architectural Support for Lock-Free Data Structures", herein Moss), in view of Oplinger et al ("Enhancing Software Reliability with Speculative Threads", herein Oplinger), further in view of Rajwar et al. ("Speculative Lock Elision: Enabling Highly Concurrent Multithreaded Execution", herein Rajwar").

6. As per Claim 1, Moss teaches: A method for executing a fail instruction to facilitate transactional execution on a processor, comprising:

wherein changes made during the transactional execution are not committed to the architectural state of the processor unless the transactional execution successfully completes (Section 2.1, it requires that the commit instruction be successful); and if the fail instruction is encountered during the transactional execution, terminating the transactional execution without committing results of the transactional execution to the architectural state of the processor (Section 2.1, see commit, abort and validate instructions), but fails to teach:

executing a start transactional execution instruction to start transactionally executing a block of instructions within a program, wherein the start transactional execution instruction marks the beginning of a block of instructions to be executed transactionally, and wherein the start transactional execution instruction specifies an action to take if transactional execution of the block of instructions fails, and wherein the action to take can include acquiring a lock on the block of instructions;

retrying the transactional execution a specified number of times, and

if the fail instruction continues to be encountered after the specified number of times, obtaining a lock for the block of instructions.

However, Oplinger teaches a transactional programming model in which his abort/fail instruction not only eliminates the speculative state, but also branches the program to an address that was previously set by a TRY instruction (Section 3.2). The advantage of having an instruction is being able to go to an error handling case in the event of an abort (see Section 3.2, the second code example, where the system jumps to an error message on an abort), giving the programmer more control over the execution and troubleshooting of his program. Given these advantages, it would have been obvious to one of ordinary skill in the art at the time the invention was made to take Moss's invention, and allow the abort instruction to specify an address to branch to in the event of the abort instruction executing.

However, neither of these references teach retrying the execution a specified number of times, and obtaining a lock if the fail instruction continues to be encountered (presumably after the specified number of times, which is not entirely clear from applicant's claim language), rather they will either try again (possibly forever without success), or will execute an abort instruction which will send the user to some kind of error handler, neither of which guarantees execution. However, Rajwar teaches a multithreaded system which implements critical sections, similar to Oplinger and Moss's systems, but when the system can not enter the critical section, it tries again several times, and if it continues to fail, will acquire a lock in order to guarantee forward progress (Page 297, Section 3.2).This is advantageous if the user does not wish to use an abort instruction, and wants or needs to execute the instruction, by forcing the system to acquire a lock, it prevents a deadlock which would cripple processor performance. Given these advantages, one of ordinary skill in the art at the time the invention was made would have been motivated to include Rajwar's teachings of acquiring a lock on the critical section after a set number of retries, in order to guarantee forward progress of the code.

7. As per Claim 2, Moss teaches: The method of claim 1, wherein terminating the transactional execution involves discarding changes made during the transactional

execution (Section 2.1).

8. As per Claim 3, Moss teaches: The method of claim 2, wherein discarding changes made during the transactional execution involves:

discarding register file changes made during the transactional execution (Section 2.1, see commit, abort, and validate instructions);

clearing load marks from cache lines (Section 3.1.2, XABORT tagged entries are set to EMPTY);

draining store buffer entries generated during transactional execution (Section 3.1.2, XABORT tagged entries are set to EMPTY, clearing out the data that was temporarily stored in them); and

clearing store marks from cache lines (Section 3.1.2, XABORT tagged entries are set to EMPTY).

9. As per Claim 4, Oplinger teaches the method of claim 1 wherein terminating the transactional execution additionally involves branching to a location specified by a corresponding start transactional execution (STE) instruction (Section 3.2).

10. As per Claim 6, Moss teaches: The method of claim 1, wherein terminating the transactional execution additionally involves attempting to re-execute the block of instructions (Section 2.2, wherein if Step 4 fails, the process repeats at step 1).

11. As per Claim 7, Moss teaches: The method of claim 1, wherein if the transactional execution of the block of instructions is successfully completed, the method further comprises:

atomically committing changes made during the transactional execution (Sections 2.0 and 2.1); and

resuming normal non-transactional execution (As Section 2.2 says, the transactional execution is intended for critical sections, which are small parts of non-transactional code blocks. Therefore, when it was finished, it would resume execution in the non-transactional code. Section 5.4 further elaborates on this, by stating that the transactions have short durations, and small data sets, meaning that it must go to non-transactional after that short duration).

12. As per Claim 8, Moss teaches: The method of claim 1, wherein potentially interfering data accesses from other processes are allowed to proceed during the transactional execution of the block of instructions (Section 1, where it is stated that "If one process is interrupted in the middle of an operation, other processes will not be prevented from operating on that object").

13. As per Claim 9, Moss teaches: The method of claim 1, wherein if an interfering data access from another process is encountered during the transactional execution, the method further comprises:

discarding changes made during the transactional execution (Section 2.1, see commit, abort, and validate instructions); and

attempting to re-execute the block of instructions (Section 2.2, see step 4).

14. As per Claim 10, Moss teaches: The method of claim 1, wherein the block of instructions to be executed transactionally comprises a critical section (Section 2.2, first paragraph).

15. As per Claim 11, Moss teaches: The method of claim 1, wherein the fail instruction is a native machine code instruction of the processor (Section 7).

16. As per Claim 12, Moss teaches: The method of claim 1, wherein the fail instruction is defined in a platform-independent programming language (Section 3.1 and 3.2, which show an example written in C).

17. As per Claim 13, Moss teaches: A computer system that supports a fail instruction to facilitate transactional execution, comprising:

a processor (inherent in a computer system that executes instructions); and

an execution mechanism within the processor (inherent in a computer system that executes instructions);

wherein changes made during the transactional execution are not committed to the architectural state of the processor unless the transactional execution successfully completes (Section 2.1, it requires that the commit instruction be successful); and

wherein if the fail instruction is encountered during the transactional execution, the execution mechanism is configured to terminate the transactional execution without committing results of the transactional execution to the architectural state of the processor (Section 2.1, see commit, abort and validate instructions), but fails to teach:

wherein the execution mechanism is configured to execute a start transactional execution instruction to transactionally execute a block of instructions within a program, wherein the start transactional execution instruction marks the beginning of a block of instructions to be executed transactionally, and wherein the start transactional execution instruction specifies an action to take if transactional execution of the block of instructions fails, and wherein the action to take can include acquiring a lock on the block of instructions;

retry the transactional execution a specified number of times, and

if the fail instruction continues to be encountered after the specified number of times, obtain a lock for the block of instructions.

However, Oplinger teaches a transactional programming model in which his abort/fail instruction not only eliminates the speculative state, but also branches the program to an address that was previously set by a TRY instruction (Section 3.2). The advantage of having an instruction is being able to go to an error handling case in the event of an abort (see Section 3.2, the second code example, where the system jumps to an error message on an abort), giving the programmer more control over the execution and troubleshooting of his program. Given these advantages, it would have been obvious to one of ordinary skill in the art at the time the invention was made to take Moss's invention, and allow the abort instruction to specify an address to branch to in the event of the abort instruction executing.

However, neither of these references teach retrying the execution a specified number of times, and obtaining a lock if the fail instruction continues to be encountered (presumably after the specified number of times, which is not entirely clear from applicant's claim language), rather they will either try again (possibly forever without success), or will execute an abort instruction which will send the user to some kind of error handler, neither of which guarantees execution. However, Rajwar teaches a multithreaded system which implements critical sections, similar to Oplinger and Moss's systems, but when the system can not enter the critical section, it tries again several times, and if it continues to fail, will acquire a lock in order to guarantee forward progress (Page 297, Section 3.2).This is advantageous if the user does not wish to use an abort instruction, and wants or needs to execute the instruction, by forcing the system to acquire a lock, it prevents a deadlock which would cripple processor performance. Given these advantages, one of ordinary skill in the art at the time the invention was made would have been motivated to include Rajwar's teachings of acquiring a lock on the critical section after a set number of retries, in order to guarantee forward progress of the code.

18. As per Claim 14, Moss teaches: The computer system of claim 13, wherein while terminating the transactional execution, the execution mechanism is configured to discard changes made during the transactional execution (Section 2.1).

19. As per Claim 15, Moss teaches: The computer system of claim 14, wherein while discarding changes made during the transactional execution, the execution mechanism is configured to:

discard register file changes made during the transactional execution (Section 2.1);

clear load marks from cache lines (Section 3.1.2, XABORT tagged entries are set to EMPTY);

drain store buffer entries generated during transactional execution (Section 3.1.2, XABORT tagged entries are set to EMPTY, clearing out the data that was temporarily stored in them); and

20. to clear store marks from cache lines (Section 3.1.2, XABORT tagged entries are set to EMPTY).

21. As per Claim 16, Oplinger teaches the computer system of claim 13, but fails to teach: wherein while terminating the transactional execution, the execution mechanism is additionally configured to branch to a location specified by a corresponding start transactional execution (STE) instruction (Section 3.2).

22. As per Claim 18, Moss teaches: The computer system of claim 13, wherein while terminating the transactional execution, the execution mechanism is additionally configured to attempt to re-execute the block of instructions (Section 2.2, wherein if Step 4 fails, the process repeats at step 1).

23. As per Claim 19, Moss teaches: The computer system of claim 13, wherein if the transactional execution of the block of instructions is successfully completed, the execution mechanism is configured to:

atomically commit changes made during the transactional execution (Sections 2.0 and 2.1); and

to resume normal non-transactional execution (As Section 2.2 says, the transactional execution is intended for critical sections, which are small parts of non-transactional code blocks. Therefore, when it was finished, it would resume execution in the non-transactional code. Section 5.4 further elaborates on this, by stating that the transactions have short durations, and small data sets,,meaning that it must go to non-transactional after that short duration).

24. As per Claim 20, Moss teaches: The computer system of claim 13, wherein the computer system is configured to allow potentially interfering data accesses from other processes to proceed during the transactional execution of the block of instructions (Section 1, where it is stated that "If one process is interrupted in the middle of an

operation, other processes will not be prevented from operating on that object").

25. As per Claim 21, Moss teaches: The computer system of claim 13, wherein if an interfering data access from another process is encountered during the transactional execution, the execution mechanism is configured to:

discard changes made during the transactional execution (Section 2.1, see commit, abort, and validate instructions); and

to attempt to re-execute the block of instructions (Section 2.2, see step 4).

26. As per Claim 22, Moss teaches: The computer system of claim 13, wherein the block of instructions to be executed transactionally comprises a critical section (Section 2.2, first paragraph).

27. As per Claim 23, Moss teaches: The computer system of claim 13, wherein the fail instruction is a native machine code instruction of the processor (Section 7).

28. As per Claim 24, Moss teaches: The computer system of claim 13, wherein the fail instruction is defined in a platform-independent programming language (Section 3.1 and 3.2, which show an example written in C).

29. As per Claim 25, Moss teaches: A computing means that supports a fail instruction to facilitate transactional execution, comprising:

a processing means (inherent in a computer that executes instructions); and

an execution means within the processing means (inherent in a computer that executes instructions);

wherein changes made during the transactional execution are not committed to the architectural state of the processor unless the transactional execution successfully completes (Section 2.1, it requires the commit instruction to successfully complete); and

wherein if the fail instruction is encountered during the transactional execution, the execution means is configured to terminate the transactional execution without committing results of the transactional execution to the architectural state of the processor (Section 2.1, see the commit, abort, and validate instructions), but fails to teach:

wherein the execution means is configured to execute a start transactional execution instruction to transactionally execute a block of instructions within a program, wherein the start transactional execution instruction marks the beginning of a block of instructions to be executed transactionally, and wherein the start transactional execution instruction specifies an action to take if transactional execution of the block of instructions fails, and wherein the action to take can include acquiring a lock on the block of instructions;

retry the transactional execution a specified number of times, and

if the fail instruction continues to be encountered after the specified number of times, obtain a lock for the block of instructions.

However, Oplinger teaches a transactional programming model in which his abort/fail instruction not only eliminates the speculative state, but also branches the program to an address that was previously set by a TRY instruction (Section 3.2). The advantage of having an instruction is being able to go to an error handling case in the event of an abort (see Section 3.2, the second code example, where the system jumps to an error message on an abort), giving the programmer more control over the execution and troubleshooting of his program. Given these advantages, it would have been obvious to one of ordinary skill in the art at the time the invention was made to take Moss's invention, and allow the abort instruction to specify an address to branch to in the event of the abort instruction executing.

However, neither of these references teach retrying the execution a specified number of times, and obtaining a lock if the fail instruction continues to be encountered (presumably after the specified number of times, which is not entirely clear from applicant's claim language), rather they will either try again (possibly forever without success), or will execute an abort instruction which will send the user to some kind of error handler, neither of which guarantees execution. However, Rajwar teaches a multithreaded system which implements critical sections, similar to Oplinger and Moss's systems, but when the system can not enter the critical section, it tries again several times, and if it continues to fail, will acquire a lock in order to guarantee forward progress (Page 297, Section 3.2).This is advantageous if the user does not wish to use an abort instruction, and wants or needs to execute the instruction, by forcing the system to acquire a lock, it prevents a deadlock which would cripple processor performance. Given these advantages, one of ordinary skill in the art at the time the invention was made would have been motivated to include Rajwar's teachings of acquiring a lock on the critical section after a set number of retries, in order to guarantee forward progress of the code.

Response to Arguments

30. Applicant has essentially argued that the current amendments overcome the rejections, as the claims currently recite a start transactional execution instruction which specifies an action to take in the event of a failure, one action possibly including acquiring a lock on the block of instructions. Examiner asserts that the start transactional instruction, taught by Oplinger does teach an action to take in the event of a failure, as it branches to an address on the event of a failure, and that address's instructions would be the action taken. While Oplinger by itself or with the other references does not teach that the action can be acquiring a lock on the block of instructions, the claim language states that the action to take *can* include acquiring a lock, meaning that does not have to be one of the actions. As Oplinger teaches that some action is taken in the event of a failure, the combination still teaches the reference, due to the language which does not necessitate that acquiring a lock is one of the actions that takes place.

31. This action has been made non-final as a result of Examiner discovering a co-pending Application sharing very similar claims to the current Application that has necessitated a Double Patenting rejection.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Robert E. Fennema whose telephone number is (571) 272-2748. The examiner can normally be reached on Monday-Friday, 8:45-6:15.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Robert E Fennema
Examiner
Art Unit 2183

RF

[illegible]

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100